

CLIENT-SIDE PAGE ELEMENT WEB-CACHING

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Ramya Karri

May 2009

© 2009

Ramya Karri

ALL RIGHTS RESERVED

SAN JOSÉ STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

CLIENT-SIDE PAGE ELEMENT WEB CACHING

by

Ramya Karri

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science

05/01/2009

Dr. Soon Tee Teoh, Department of Computer Science

05/01/2009

Dr. Tsau Young Lin, Department of Computer Science

05/01/2009

ABSTRACT

CLIENT-SIDE PAGE ELEMENT WEB CACHING

by Ramya Karri

When a user explores different web pages in a given website, the website typically sends the entire requested page even if only a portion of the page was different to the current page. That is, two pages on a given website might share elements on the page like search bar, left bar, navigation controls, advertisements, etc., but this information is retransmitted. Most of the users spent their time on the front-end while downloading all the components in the page. Nowadays, server-side caching of page elements is often done using tools like memcached. The aim of my project is to explore element web page caching on the client-side. That is, our goal is to develop a system that caches the most common html parts of web pages in the website and reuses them in the further web pages reducing the transmission data. This effect probably is currently attainable using frames or object tags; however, the actual UI meaning of these tags is different than one integrated HTML file and so could cause usability issues, therefore, we want to explore solutions which are transparent to the end user -- the solution must behave just like a single fixed web page. In order to explore the advantage of having client-side caching and determine the effect on the response time, we made our server set-up as realistic as possible. So Squid, a front-end load balance, was used when we tested our client-side caching.

ACKNOWLEDGEMENTS

I would like to thank my husband for his support in countless ways on countless occasions. I would also like to thank my advisor, Dr. Chris Pollett, for his suggestions and guidance, and my committee members, Dr. Soon Tee Teoh and Dr. Tsau Young Lin, for their time and effort.

Table of Contents

1	Introduction.....	1
2	Tools Used.....	3
2.1	Apache HTTP Server2.2.....	3
2.2	PHP 5.2.8.8.....	3
2.3	CakePHP.....	4
2.4	YSlow.....	4
2.5	iMacros.....	5
2.6	Squid Web Proxy Server.....	5
3	Preliminary Work.....	6
3.1	Develop course listing website.....	6
3.2	Balance load between servers.....	7
3.3	Determine the advantage of caching text.....	8
3.4	Analyze Firefox caching mechanism.....	9
4	Design.....	10
4.1	Server.....	12
4.2	Client.....	13
5	Implementation and Results.....	14
5.1	How to install our project.....	14
5.2	Deliverable 1: Creating the caching mechanism.....	20
5.2.1	Motivation.....	20
5.2.2	Goal.....	21

5.2.3	Implementation and Results.....	21
5.2.4	Remarks.....	26
5.3	Deliverable 2: Testing the caching mechanism	29
5.3.1	Motivation	29
5.3.2	Goal	30
5.3.3	Implementation and Results.....	30
5.3.4	Remarks.....	53
6	Conclusion	55
7	References.....	56

List of Figures

Figure 1: Course listing website	6
Figure 2: List of courses	7
Figure 3: Caching Experiment	8
Figure 4: Model View Controller Pattern	11
Figure 5: Directory Structure of Apache	16
Figure 6: Project folder	17
Figure 7: Directory Structure of Squid	18
Figure 8: Configure browser with Squid	19
Figure 9: Test web page with Javascript on.....	32
Figure 10: Test web page when Javascript is off.....	33
Figure 11: Results obtained using Firefox browser.....	50
Figure 12: Results obtained using Internet Explorer	51
Figure 13: Results obtained using Opera	52
Figure 14: Results obtained using Safari browser	53

1 Introduction

Due to the tremendous increase in the use of internet, there is a lot of ongoing research in the field of web page caching both at the server and client-side. Despite all this research, 80% of end-user response time is spent on the front-end while downloading all the components in the page such as images, style sheets, scripts, Flash, etc (“Best Practices for Speeding Up Your Web Site,” 2008).

The time taken by the web page to load is defined as the response time of the web page. Currently, most websites share common elements in their web pages. However, when the user explores web pages of the same website, even when the pages have common portions, such as help boxes, navigational controls, extra menus, login forms, with the current page, the website retransmits the entire requested page to the user. Our goal was to reduce this response time of the web pages at least by 20% by caching these most common elements and reusing them in the further requests instead of retransmitting it from the server on every request.

Therefore, we developed the methodology and the apparatus of a new form of web page caching at the client-side that caches the most common parts of the website at the client-side and reuses them in the further requests reducing the total download time of the web page. In order to find the advantage of having a client-side cache, we made the server as realistic as possible. Therefore, Squid, a front-end accelerator was used while testing our caching technique.

This project is mainly divided into two deliverables: first, developing the caching mechanism; second, testing the caching mechanism. Deliverable 1 consisted of designing and developing a new caching technique which is transparent to the user. Our caching mechanism is designed such that it works in harmony with the web browsers caching mechanism to produce our desired output, i.e., to get a reduction in the response time of the web pages. Deliverable 2 involved testing the caching technique and is further divided into the following sub tasks: developing the website using our new caching technique and also using the conventional method, measuring their response times, and comparing their response times.

The paper gives a clear idea as to how the experiment was conducted. It is structured as follows: Section 2 describes the various tools used to develop the caching mechanism. Section 3 gives the details of the preliminary work done. Section 4 describes the high level design, implementation of the project and results are discussed in Section 5; Section 6 gives us the conclusion.

2 Tools Used

The majority of the work done in this project is associated with the World Wide Web. It involves developing a new form of web page caching and performing tests to measure the performance of the new caching system, therefore following tools have been used.

2.1 Apache HTTP Server 2.2

The main purpose of using Apache HTTP server in our project is to host our web application. The web pages requested from our website will be served by our Apache server.

Apache is a freely available web server primarily used to serve both static content and dynamic Web pages on the World Wide Web. Web servers are computers on the internet that host websites and serve pages to users upon request. When user requests a web page on a web browser, this redirects the page to the respective web server. The web server fetches the requested web page and sends it to your browser.

2.2 PHP 5.2.8.8

PHP is a widely-used general-purpose scripting language that is especially suited for web development (“PHP,” 2008). It is mainly used for developing dynamic content from web servers to clients. PHP consists of native API’s to Apache HTTP Server and when Apache gets a request for a web page with PHP, it calls the PHP interpreter to generate HTML and then this HTML page is returned to the client. As we have used

Apache HTTP Server as the web server, we have used PHP to dynamically generate web content for our application.

2.3 *CakePHP*

We have developed our caching mechanism as an extension to the CakePHP caching mechanism. It provides a rapid development framework for developing PHP applications. It uses Model-View-Controller (MVC) design pattern, which reduces the development costs and helps developers write less code (“CakePHP,” 2008). Model represents the tables in the database used for the application. Controller deals with the logic of the application and View represents the html part needed to display the web pages. It makes it easier to modify the user interface design without making changes to the business logic and vice versa. Along with MVC, CakePHP offers many features such as helpers for AJAX, Javascripts, HTML Forms, caching, which are very useful for developing web pages more efficiently with few lines of code.

2.4 *YSlow*

As our goal is to reduce the response time of the web pages, in order to measure the response time of the web pages created using our new caching mechanism, we used YSlow to record the response times of the web pages in our web application.

Yahoo’s Exceptional Performance team has identified a number of best practices for making web pages fast and YSlow is a Firefox add-on that analyzes web pages and tells us why they are slow based on the rules for high performance web sites (“Speed up your web pages with YSlow,” 2009).

2.5 iMacros

iMacros is a HTML based macro recorder used for web automation, web test, and data extraction. In order to observe how our caching system works in Internet Explorer, we needed a tool that measures the response time of the web pages when requested using Internet Explorer. As there is no YSlow extension for Internet Explorer, we have used iMacros in our project to record the response times of the web pages.

2.6 Squid Web Proxy Server

In order to make the server as realistic as possible we used Squid while testing our caching mechanism. Squid is a caching proxy for the web which reduces bandwidth and improves response times by caching and reusing frequently requested web pages.

A caching proxy server sits between a web browser and a web server. It keeps local copies of frequently requested resources and intercepts the requests made to the original server to see if it can serve the requests itself, otherwise it forwards them to the original server.

3 Preliminary Work

Most of the initial research about the project was done during my CS297, Preparation for writing project. During my research phase, I have learned about all the tools required for this project and have also done few experiments using them. I have categorized my work into four deliverables: first, develop course listing website; second, balance load between servers; third, determine the advantage of caching text; fourth, analyze Firefox caching mechanism.

3.1 Develop course listing website

PHP and CakePHP being the main tools used in creating our caching mechanism, this deliverable was mainly prepared to discover the main features in both of the tools so as to use them to build our new caching technique. To get hands-on experience with these technologies, I created a course listing website (Fig. 1) that displays the courses offered by the computer science department in a particular year and semester.

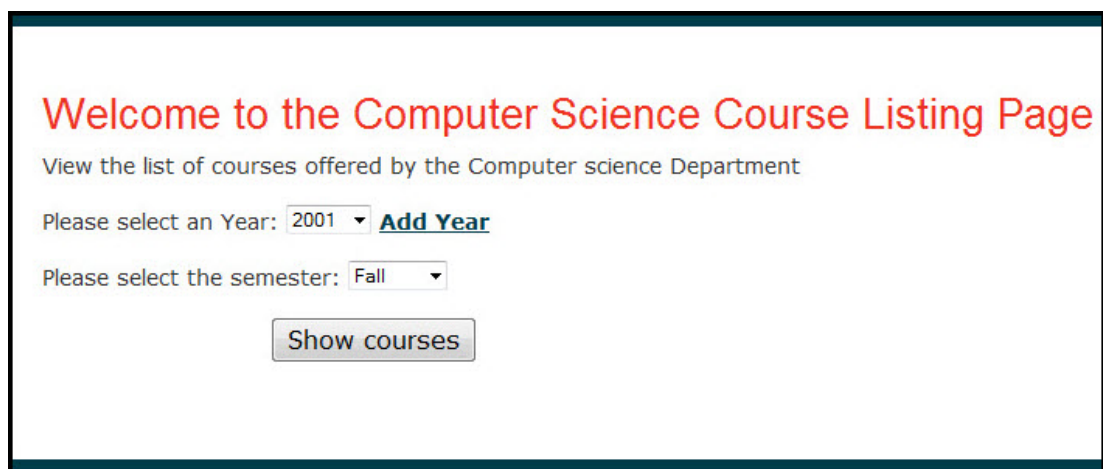
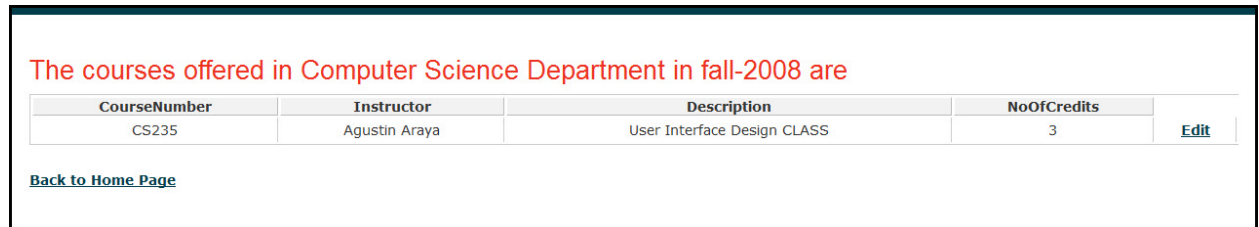


Figure 1: Course listing website

In order to view the list of courses, we should select the year and semester in figure 1 and click on Show courses button. Once the button is clicked the following screen (Fig. 2) appears displaying the list of courses offered in the selected year and semester.



CourseNumber	Instructor	Description	NoOfCredits	
CS235	Agustin Araya	User Interface Design CLASS	3	Edit

[Back to Home Page](#)

Figure 2: List of courses

We can also add new courses to the existing list and edit the details of the courses. The main idea was to explore all the CRUD operations for the database such as read from database, write into the database, offered in CakePHP. Doing this experiment helped in better understanding as to how to develop an application very efficiently using PHP and CakePHP.

3.2 Balance load between servers

Currently, proxy servers are being used by hundreds of internet providers worldwide to provide their users with the best possible web access. Therefore, the main goal of this deliverable was to understand Squid proxy server and discover how to setup a LAN (Local Area Network) consisting of a web server, a proxy server, and a web client. This LAN setup will be used further in the experiment when we test our caching technique. Once the LAN was setup, we extended this deliverable to configure Squid to balance the load between two web servers, i.e., when requests are made to the Squid it checks if it has

a copy in its own cache, otherwise it balances the load between two web servers by forwarding the requests to the two web servers in a round robin fashion. This involves adding one more server to the existing LAN setup. We developed two identical servers, each having the same set of pages in them and we configured the setup files in Squid such that it can perform load balancing between the two servers.

3.3 Determine the advantage of caching text

This deliverable involves conducting experiments using web pages to understand the advantage of caching text and determine how much reduction in the response time can be obtained. To accomplish this task, I created several web pages with images and only text; compared the total download times of pages having images with pages having only text. Below graph was obtained, which shows that there might be minor advantage i.e. reduction in response time can be achieved if the text is cached.

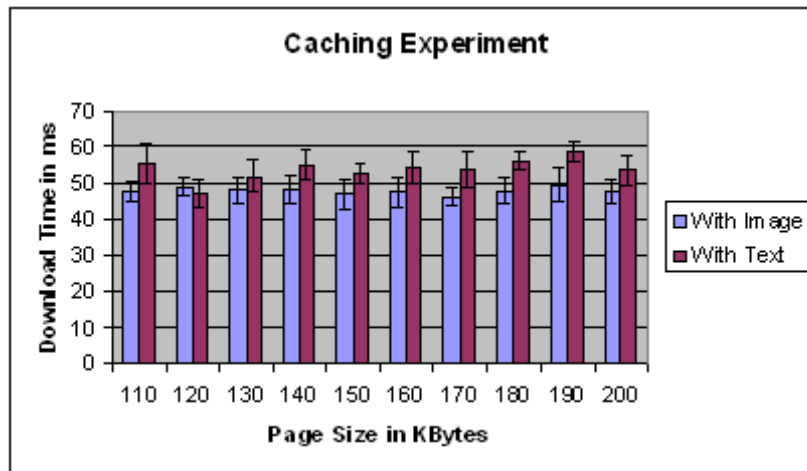


Figure 3: Caching Experiment

3.4 Analyze Firefox caching mechanism

The main purpose of this deliverable is to analyze the Firefox caching technique. This experiment mainly involved creating web pages having an image and script tag to understand how Firefox caches these objects in its cache. This experiment was conducted to give idea as to how we can use Firefox caching technique to cache the most common elements in our website.

Since CakePHP is an open source tool, I got an opportunity to study its source code which helped to get a better understanding as how the new caching technique can be developed.

4 Design

In CakePHP, if there are small blocks of presentation code that needs to be repeated from page to page, sometimes in different places in the same layout, then these blocks are stored as Element objects. In order to display these elements in the view, CakePHP uses a powerful functionality called the element function which simply works by passing the element name as the parameter.

However, these common portions need to be cached on the client-side. Therefore, we have researched about various other tools to find which can help us achieve this. The Firefox browser revealed that Firefox browser can cache external Javascripts. Therefore, we thought it would be a new direction to combine CakePHP element function and the Firefox caching technique to build our new caching technique.

The idea was to send these CakePHP elements as external Javascripts to the client's browser so that they get cached in the browser, and in the further requests are served from browser by fetching them from the browsers cache without getting transmitted from the server thereby reducing the response time.

The design pattern used to develop our project is Model View Controller. It can be shown as below.

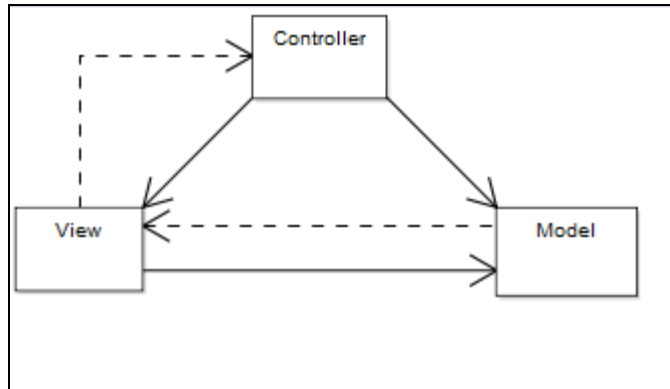


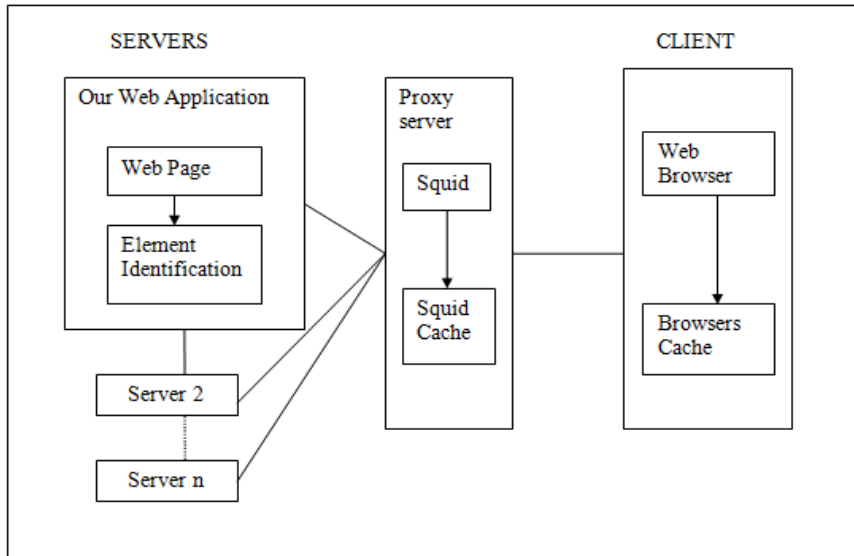
Figure 4: Model View Controller Pattern

Retrieved April 02, 2009, from

<http://upload.wikimedia.org/wikipedia/commons/2/2e/ModelViewControllerDiagram.svg>

This pattern isolates the business logic with the presentation logic. Therefore, we can change the look of your web page without having to change the underlying business code. We followed this pattern while developing our web application; therefore we can modify the web page without having to change the code used to connect the databases or change the database at the back-end without having to change the presentation logic.

As the project involves developing a new caching mechanism used to develop a website, two main components are required to achieve a working system, first, a web server, which hosts our website; second, a web browser that makes requests to the web server. But in order to make the server as realistic as possible we also used a proxy server. The high level architecture of the project is as shown below.



4.1 Server

We used the Apache HTTP server to host our website. As our new caching mechanism would be combination of CakePHP's element function and the Firefox browser caching function, we had to develop a procedure that helps us convert the CakePHP's elements into external Javascripts so that they can be cached in the browsers cache and later is reused in the further requests.

Squid is a proxy server used to serve web pages to the client on behalf of the web server. When Squid is requested for a web page, it first checks if it has a copy in its own cache that it can serve; otherwise, it forwards the request to the original web server. The web server serves the request and sends the appropriate web page to the Squid. Squid then stores a copy in its own cache and then serves it to the web browser.

4.2 Client

The system used to request web pages is called a client and is typically a web browser, or may be a web crawler. Once the pages are requested, the browser checks to see if the page exists in its own cache, otherwise it redirects the request to Squid proxy server. Squid checks to see if it can serve the page from its own cache, if not it will redirect the request the page from the web server.

The main browsers used to test our caching mechanism are Firefox and Internet Explorer. As Firefox and Internet Explorer browsers cache external Javascripts, we wanted to take advantage of them to store the common elements on the client-side, the procedure that we developed sends the CakePHP elements as external Javascripts to the web browser so that these are cached at the client-side. In further requests these elements are served from the browsers cache instead of getting retransmitted from the server and hence reduction in response time can be obtained.

5 Implementation and Results

This section illustrates how the apparatus of the new caching technique was built. It clearly explains the procedure followed to test the caching mechanism and also demonstrates the results obtained.

As discussed in the design, since current web browsers cache external Javascripts, to reduce the response time of a web page we need to serve the most common elements as external Javascripts to the client's browser so that they get cached in the browser and will be reused in the further requests rather than retransmitting them from the original server.

In order to serve the most common elements as external Javascripts to the client browser, we needed a procedure that identifies the most common elements in the website and converts them as external Javascripts. We also needed a test website to test the performance of our caching mechanism. Therefore, the project was mainly divided into two deliverables, first, creating the caching mechanism; second, testing the caching mechanism. Prior to discussing the details of these deliverables let us see how to install the project.

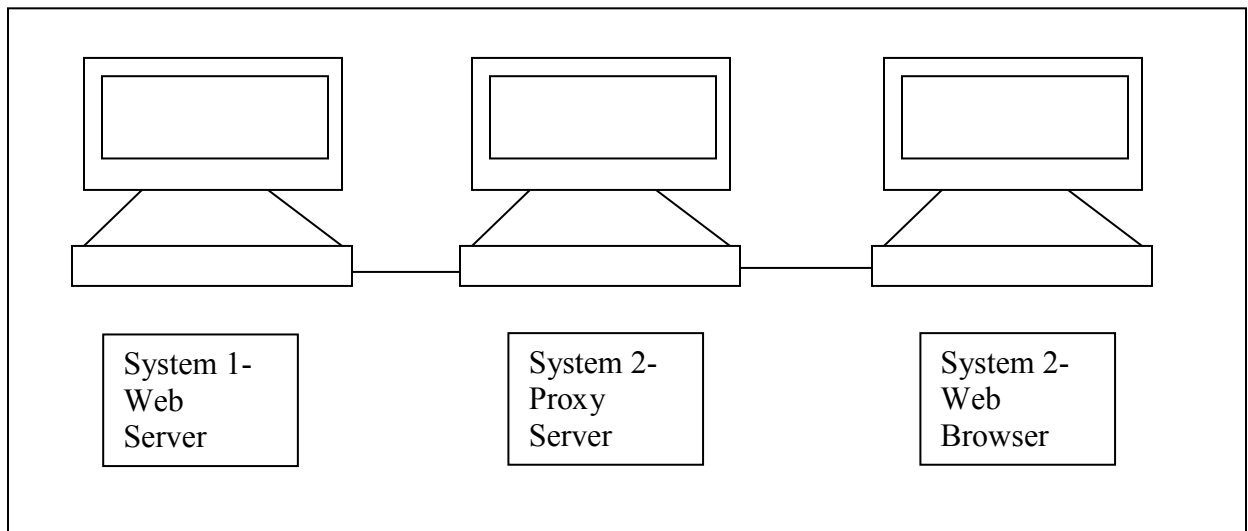
5.1 How to install our project

This section gives a very clear idea as to how to install our project. Our project is available for download in the internet from my advisor's website under Master's Research student's link. You can also directly go to the project folder using the link,

<http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Fall08/ramya/project.zip>. The following are steps needed to install the project.

Step 1: Download the zip file and save it on the desktop.

Step 2: Create a LAN setup. Our LAN setup consists of a web server, a web client and a proxy server, i.e., we need three systems running Windows Operating systems.



Step 3: Once we have all the three systems, connect them in a LAN. Try to ping the systems from other two systems. If the systems cannot be pinged from one another, try switching off the firewalls and try again.

Step 4: Once the LAN system is established. Take the system 1 and make it as a web server by installing Apache HTTP server on it. Once Apache HTTP server is installed you can find the Apache folder in your C:\ drive. The directory structure of Apache is as shown below.

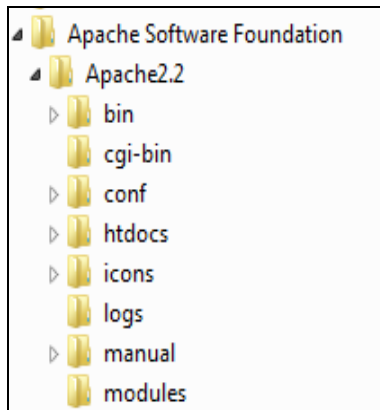


Figure 5: Directory Structure of Apache

Step 5: Now install the latest stable version of PHP from PHP official website, <http://www.php.net/>. After PHP is installed, configure PHP to work with Apache HTTP server.

Step 6: Now unzip our project file, which was initially placed on the desktop and put in the htdocs folder of Apache, as shown below.

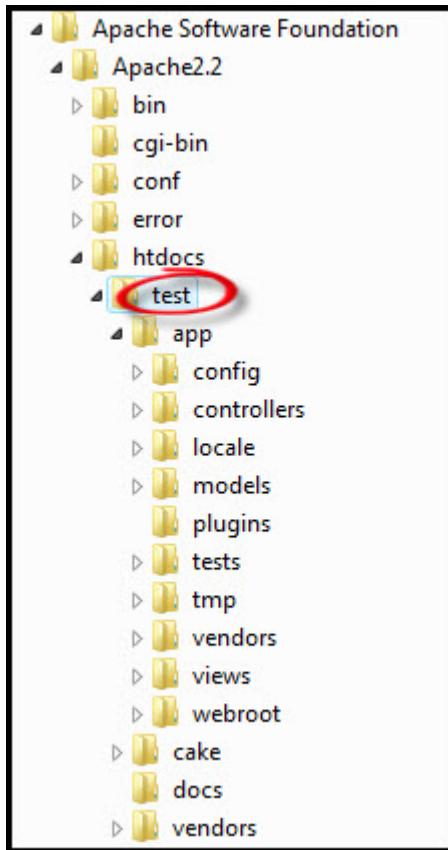


Figure 6: Project folder

Step 6: After you place the project folder inside the Apache htdocs folder, restart the Apache HTTP server process.

Step 7: Now we need to configure the Apache HTTP server to work with our project, i.e., open the httpd file of Apache which resides in /conf/ folder. Search for Document Root in that file and replace the path of the DocumentRoot with the path to our project, i.e., add /test/ to the existing path of the DocumentRoot.

Step 8: Now setup the proxy server. I.e., install Squid proxy server in System 2. After installation you can find the Squid folder in C:\ drive as shown below.

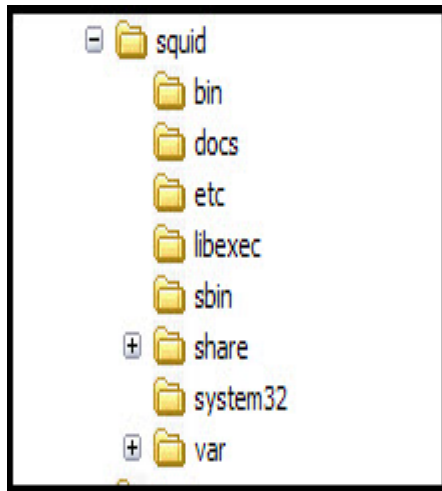


Figure 7: Directory Structure of Squid

Step 9: Configure the Squid proxy server to allow request to our web server, this can be done by modifying the squid.conf file.

Step 10: Now check if the Client system consists of a Firefox web browser, otherwise please install Firefox web browser.

Step 11: Now configure the Firefox web browser to use Squid as its proxy server. This can be done by opening Firefox browser > Go to Tools > Options > Advanced > Click on Network tab > Click Settings button in the connection tab > Select the radio button use proxy server and Enter address of the proxy server for Address and 3128 for Port.

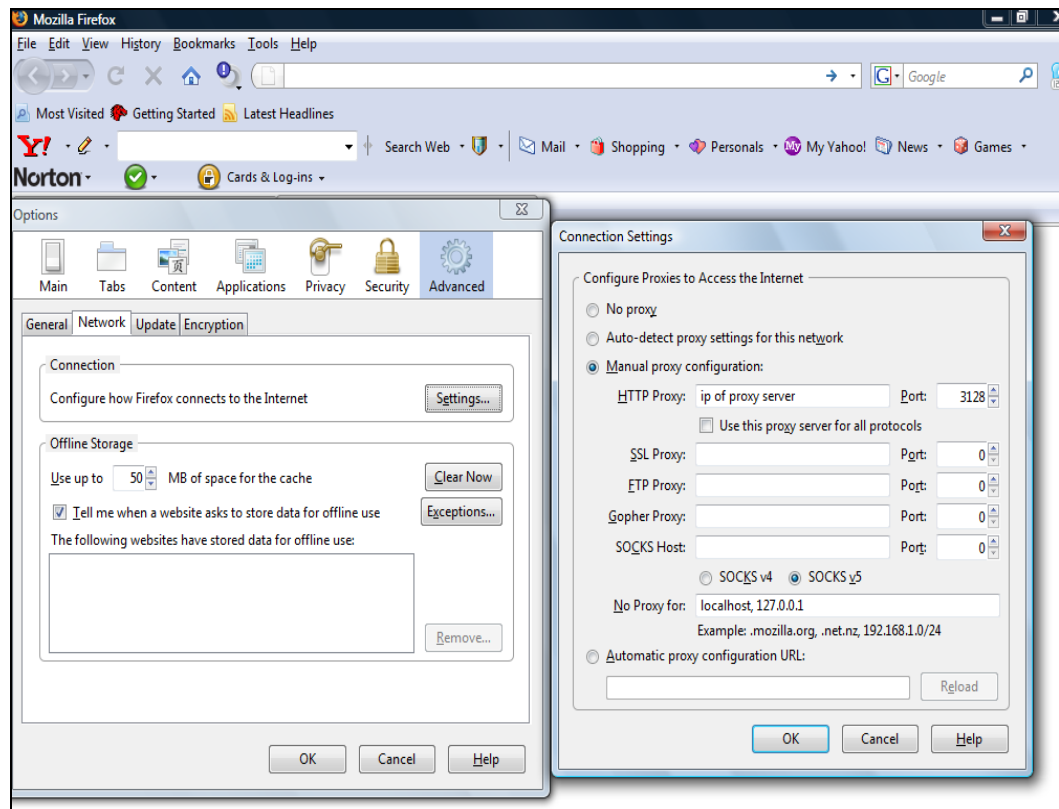


Figure 8: Configure browser with Squid

Step 12: In order to test using Firefox, install YSlow, a Firefox add-on to view the response times of the web pages. If you want to use Internet Explorer, install iMacros recorder and record the response times of the web pages using it.

Step 13: Now that all the required the tools are installed, proxy server is configured to allow requests for our main server, and browser is configured to send requests to Squid proxy server we are ready to request our web pages. This can be done by using the following URI.

`http://<ip.of.the server>//testscripts/homepage`

The URI for the remaining web pages are,

For page 2, `http://<ip.of.the server>/testscripts/page2`

For page 3, `http://<ip.of.the server>/testscripts/page3`

For page 4, `http://<ip.of.the server>/testscripts/page4`

Using the above steps we can successfully install the project and run the four test pages in a LAN environment. We can also record the response times of the web pages using YSlow plug-in, we have installed in the Firefox web browser.

5.2 Deliverable 1: Creating the caching mechanism

5.2.1 Motivation

To obtain a reduction in the response time, we need to use a caching mechanism. Currently, web browsers are equipped with a cache that helps us serve web pages faster. However, they are not equipped with a mechanism that identifies the repeated portions of the webpage. CakePHP also provides us with a platform where the web page can be created very efficiently with less code. When we have common portions in the web page that need to be repeated on several web pages or need to appear at different positions on the same page, these common portions are saved as elements and are rendered as sub-views on the main view, i.e., main web page using element function. But CakePHP does not provide a method that converts these Element object into Javascript. Therefore, our new caching mechanism was developed as an add-on to the CakePHP element function in combination with the Firefox caching mechanism to produce the desired result, i.e., reduction in response time for the web page.

5.2.2 Goal

The goal of this deliverable is to create a new web caching technique that renders common portions of website as Javascripts to the web browsers so that they can be cached in the browsers cache and reused, thereby reducing the transmission time on further requests. This caching technique was developed to be compatible with both the Firefox caching and CakePHP caching and is transparent to the end user, i.e., the user will not notice the underlying caching technique when the webpage is displayed in the web browser.

5.2.3 Implementation and Results

The procedure needed to develop the new caching mechanism can be divided into sub tasks as follows: identify the most common portions of the web site, render these common portions as Javascripts to the web browser, and cache these common portions at the client.

5.2.3.1 Cache common portions of the website at the client

As the main aim of this project is to reduce the response time of the web page by caching the common parts of the website at the client-side, we have explored several caching features available in the web browsers. During our research we noticed that both Firefox and Internet Explorer browsers cache external Javascripts. Therefore, we decided to send these common portions of the web page as external Javascripts which will be cached in the web browser and are reused from the cache while serving the web page in the further requests.

5.2.3.2 Identify the most common portions of the web site

Identification of the most common portions of the website should be done by the developer who is developing the website. As the developer of the website knows as to how and where the content should be placed in the web page, he will have complete knowledge as to which portions should appear commonly across the pages.

By taking advantage of the CakePHP's element feature, instead of having to write the HTML portions for common elements on every webpage, the developer stores the html code for that portion as an element and when required the element can be included in the webpage using element function by passing the element name as the parameter.

The prototypical use of CakePHP's element method is as follows.

```
<?php echo $_this->element('_helpbox'); ?>
```

The elements name is passed as the parameter, say for example helpbox is the element name, this element function renders the content of the helpbox and PHP's echo function outputs the content of the helpbox. Using this approach will reduce the coding required to create the webpage.

5.2.3.3 Render common portions as Javascripts to the web browser

Even though using CakePHP's element function helps us identify the most common portions of the website and store them as elements, in order to cache them in the web browser we need to render these elements as external Javascripts to the web browser.

In order to determine how to convert the common elements as external Javascripts we have researched how CakePHP's view caching mechanism works. Using CakePHP caching technique we can cache the commonly requested views and elements. However, these views and elements are only stored as files. Since we have to render these elements as Javascripts in order to cache them at the client, we have created a procedure which is an extension to CakePHP's element function.

This prototypical use of the new element method is shown as follows,

```
element($name, $params = array(), $loadHelpers = false, $js)
```

The input parameters for the method are name of the element, represented with \$name, \$params takes the options for the element along with the data to the element, \$loadHelpers are used to import the helpers, and \$js is set to 0 if Javascript of the web browsers is off or is set to 1 if the web browsers Javascript is on. It is similar to regular CakePHP element function with an additional parameter \$js which indicates the status of the Javascript.

Having this variable is very important because our goal of reducing the response time of the web page by caching the most common elements of the website as external Javascripts in the web browser is attainable only when browsers Javascript is on, otherwise the elements are rendered using the conventional way , i.e., using regular element function as shown in 5.1.3.2. Therefore, it is important to check the status of the browsers Javascript.

If the Javascript is off, the request is forwarded to the CakePHP base element function; otherwise our new procedure will create an external Javascript of the element and inserts an empty div tag in the elements place. This external Javascript will replace the empty div tag with the actual content of the element. In this manner we rendered the element as a Javascript to the web browser.

To dynamically replace the contents of the div tag, JQuery was used. JQuery is a fast and concise Javascript library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. JQuery's html function is used to replace the content of the div tag dynamically, and the syntax is shown below.

```
$("#div").html("<span class='red'>Hello <b>Again</b></span>");
```

The above statement replaces the contents of the div tag with `Hello Again`.

When the browser displays the page to the user all the div tags representing the common portions, i.e., the elements will be replaced by their actual content using the external Javascripts. Since external Javascripts are cached at the browser, in the further requests the elements are served from the browsers cache instead of retransmitting them from the actual server, hence reducing the response time of the web page.

Let us consider an example that illustrates how an element is rendered as Javascript using our new caching mechanism. Suppose we want to use a search bar in all

the pages in our website; first we have to create an element for the search bar. Elements live in CakePHP `/app/views/elements/` folder, and have the `.ctp` filename extension. Once the element is created, include it in the required web pages using the new element function as follows.

```
<?php echo AddonsView::element('searchBar', null, null, $status); ?>
```

`AddonsView` is the name of the view file which is an extension to the CakePHP view file. It consists of the new element function which is an add-on to CakePHP element function. The input parameters to the function are name of the elementsearch bar, second and third parameters represent additional options for the element and helpers required to be imported for the element; these are null because we are not sending any options for the element and we do not need to import any helpers. The `$status` variable represents the status of the browsers Javascript; if the Javascript is on the value is 1, otherwise the value is 0.

When the web page is requested from the browser, CakePHP renders all the components of the page; if the Javascript is on, our element function includes an empty div tag, with element name as the id, in the web page, i.e., the following div tag appears in the web page

```
<div id="searchBar"></div>
```

An external Javascript is created using JQuery html function to replace the contents of the div with actual element content at the browser. The contents of the Javascript are shown below.

```
$("#searchBar").html("----html code of seachBar----");
```

In this manner, using our new caching mechanism, we have served the element as an external Javascript to the web browser so that it can be cached in the browsers cache and can be used directly from the cache instead of retransmitting it from the server.

5.2.4 Remarks

Working with this deliverable gave me a very clear understanding of the working of the Apache web server, the Firefox web browser, and CakePHP. However, there were many challenges that we have overcome during the development of the caching technique and they are given as follows.

5.2.4.1 Interpret the source code of CakePHP and Apache

This was my first application wherein I had to interpret the source code of the tools I am using. Initially it consumed a lot of the time but soon with the help of my advisor and by using various internet resources I quickly learnt how to read the source code of the tools which helped me a lot in the later phase of my project.

5.2.4.2 Deprecated renderElement function

Initially, when I was researching the features of CakePHP, I found this renderElement function which is used in CakePHP to render the common portions of the web page; I started my experiments using renderElement function. However, later I found that this renderElement got deprecated with element function and had to change my code.

5.2.4.3 Test the client browser

Testing the client browser mainly involved tests to see if the browsers Javascript is on or off. Though most of the browsers available today support Javascript, yet considering those old browsers in our mind, we wanted to make sure that our web page is displayed properly even when the web browsers Javascript is off. Therefore, in our web pages when client's Javascript is off, we used a noscript tag to redirect the web page. However, having a noscript tag in head tag did not satisfy HTML 4.0.1 Strict standard, so we had to explore new ways to make this work. We were very pleased to see that HTML5 standard supports it therefore we used HTML5 as the document type of our website.

5.2.4.4 Subclass of the view class

The CakePHP mainly consists of two main folders; an app folder where we can create the web application and a cake folder in which all the main core files exist. If we need to upgrade to a different version of CakePHP, we just need to replace the cake folder with the new cake folder. As the new caching mechanism requires creating an

extension to the CakePHP element function, which resides in the view file under the cake folder; modifying the cake folder is not recommended as the code will be lost if the CakePHP version is updated, therefore I created a subclass of the view file which contained the new element function in the app folder.

5.2.4.5 Placement of the Javascript file

In CakePHP when we use Javascript helper to create Javascripts, CakePHP typically searches for the Javascripts in its /app/webroot/js/ folder. But in our technique when we dynamically create the external Javascript of the element, we store them in CakePHP cache folder, i.e., in /app/tmp/cache/js/ folder. It was very challenging to make CakePHP to search for Javascript files in cache folder instead of webroot folder.

To achieve this, we changed the url such that if it is a Javascript created using our caching technique then the url for the Javascript will look like `http://<ip of the server>/Jscached/<name of the Javascript>`. We have modified the .htaccess file in the app folder so as to redirect the Apache to look for the Javascripts in the CakePHP cache folder. This helped us solve this problem and now we can successfully get access to these Javascripts from the cache folder.

5.2.4.6 Multiple Lines in Javascript file

Element consists of html code of the common portion that we wanted to display on all of our web pages. However, html code is written across multiple lines. But in our caching mechanism when we convert these elements into external Javascripts using JQuery html function, we encountered a Javascript error as the element consists of

multiple lines. However, using PHP's `preg_replace` function we have overcome this error by making the multiple lines appear as single statement to the Javascript.

5.2.4.7 Handle Special characters in the Javascript

In Javascript, in order to retain the actual meaning of special characters we needed to use an escape symbol before the special character, also called as escaping special characters. When we were manually creating the Javascript, the developer could handle the escaping but in our new caching mechanism; when the Javascripts are dynamically built this has to be taken care so that the Javascript does not break. Therefore, we used PHP's `addslashes` function to escape all the special characters in the element before the external Javascript is built, in this manner we not only retained the actual meaning of the special characters but also made sure that Javascript does not break due to presence of special characters.

5.3 Deliverable 2: Testing the caching mechanism

5.3.1 Motivation

We have successfully created the new caching technique but in order to test our caching mechanism, we need to develop a website that helps us determine the level of improvement in the response time of the web pages using our new technique.

5.3.2 Goal

The main goal of this deliverable was to create a website that represents a typical website most commonly visited in the internet such as Google search engine, Techsmith, Yahoo, or Greatandhra. It also involves analyzing how many common portions are being shared in these websites. This will help us understand how much response time can be saved using our caching technique. This deliverable was extended to analyzing how our caching system works with various web browsers such as Firefox, Internet Explorer, Safari, and Opera.

5.3.3 Implementation and Results

In order to make the testing as realistic as possible, we created a LAN setup, consisting of a Apache web server in which our website resides, a Squid proxy server was used as a caching proxy server which serves the requests on behalf of the web server, and a Client which consists of a Firefox web browser used to request the web pages from the server.

However, before creating a website we needed to make sure that our caching technique was actually working. Therefore, testing was also divided into subtasks, first, creating a web page with single element; second, developing the main website; third, recording the response times of the web pages; fourth, comparing the response times of the pages.

5.3.3.1 Creating a web page with single element

The purpose of this test was to make sure that all the components in our caching technique were working properly, i.e., when an element is included in our web page using our new element function, on getting a request from the browser, the element must be rendered as an external Javascript to the browser so as to get cached in it for further reuse. We also need to make sure that our external Javascript, used to replace our element div tag with the actual content of the element at the browser, is safe. To ensure this we have tested our web page by passing different values in the element.

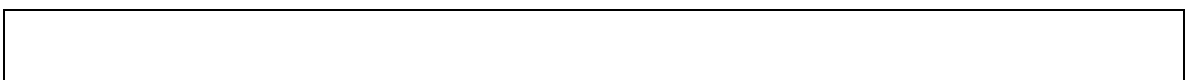
We have used a single web page containing a single element to perform this task.

The view of the webpage is as follows.

```
<?php echo $page_heading; ?>
<?php
    include('addons.php');
    echo AddonsView::element('divcontent1',array("cache"=>"5 min"), null,
$status);
?>
```

This web page consists of a page heading whose value will be passed from the CakePHP controller, a single element, called divcontent1. As can be noticed we have included the element in our view using our new Element function.

The html code of divcontent1 is given below.



```
I am in element <a href=""> welcome to this page; I am a student of San Jose State
University; <br/></a> <p>hi</p>
```

When testing this web page, we have ensured the web page is working both when the browsers Javascript is on and off. When the Javascript is on, an empty div tag having the id as divcontent1 will be added to the view page, and an external Javascript will be created that will replace the contents of the div tag with the actual element contents at the web browser. The following is the output of the web page when Javascript is on.

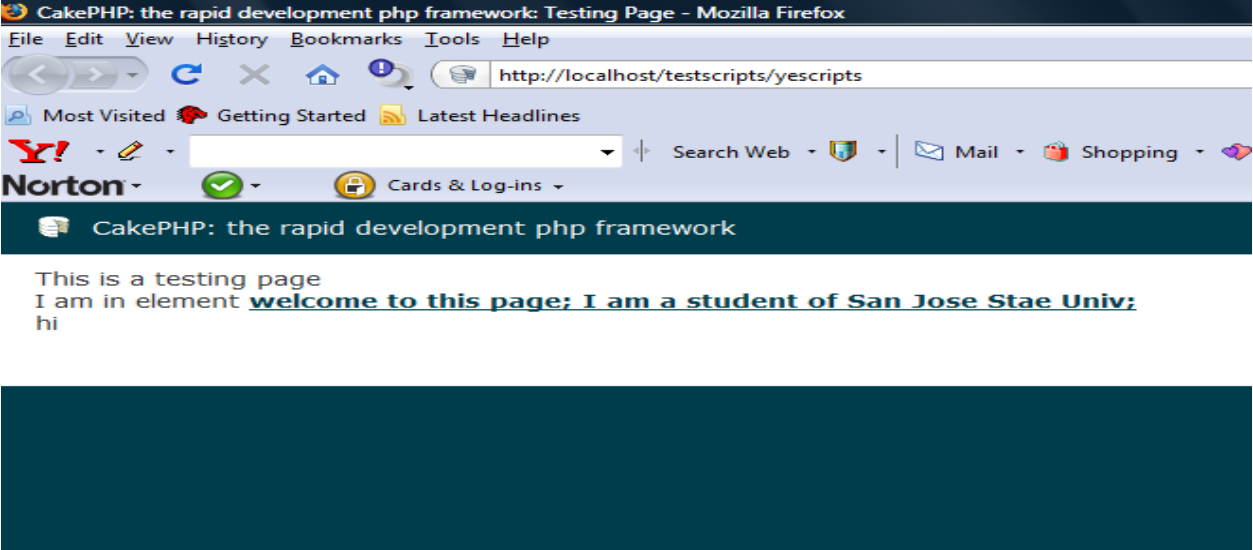


Figure 9: Test web page with Javascript on

When the Javascript of the browser is off, the element request will be forwarded to regular CakePHP element function and therefore the actual content of the element will be included in the web page. The following is the output of the web page when Javascript is off.

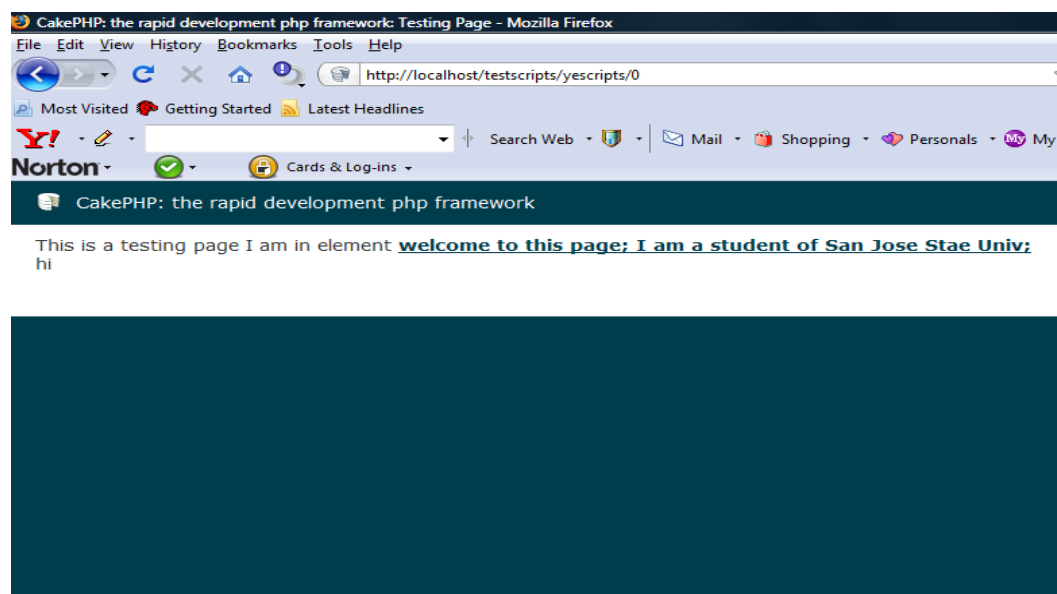


Figure 10: Test web page when Javascript is off

Hence, the web page is displayed properly both when the browsers Javascript is on or off. Our new caching mechanism successfully rendered the element as an external Javascript to the web browser. However, there is a slight difference in the web pages shown above. In the web page having Javascript on (Fig. 5), the element is displayed on a new line; whereas in the web page having Javascript off (Fig. 6), the element is displayed in the first line immediately after the text.

In the cases where browsers Javascript are on, the element has been converted into an empty div tag and an external Javascript is created. This external Javascript will replace the contents of the div tag with the actual contents of the element. Therefore, when the div tag is replaced with the actual content in the view, the element is embedded within div tags. Hence, the element is displayed in the new line whereas when the

Javascript is off, only the actual content of the element is added to the view it does not have any div tags, so it gets displayed in the first line immediately after the text.

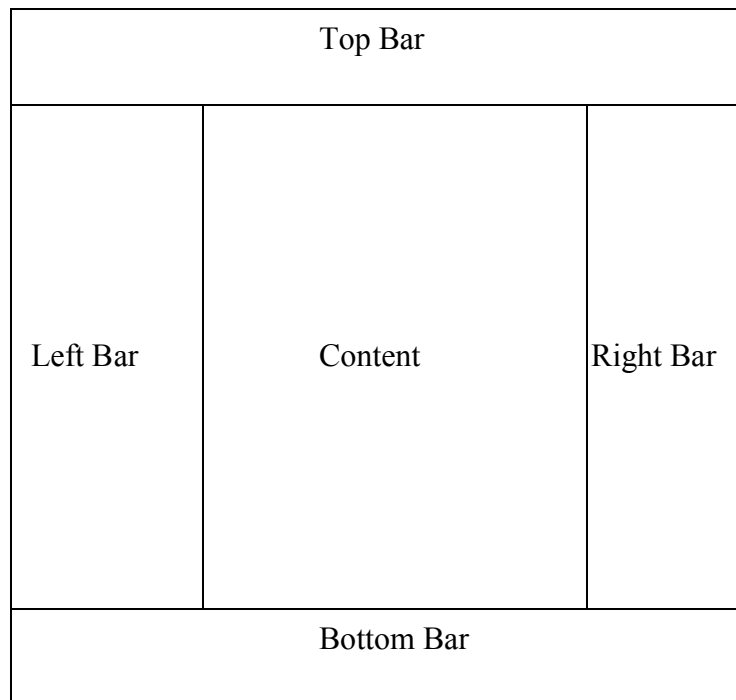
Performing this task helped us confirm that our caching mechanism was working properly. We also made sure that this web page works properly even when the element contains multiple lines of code and when it has special characters. Now that our caching mechanism worked properly, our next task was to create a website that will be used to test our caching system and determine if any reduction in response time has been obtained.

5.3.3.2 Developing the main website

As our main aim was to reduce the response time of the website by caching the most common portions in it, we developed a website that shares a few common elements across all its web pages. Testing was done in several phases.

Four web pages were used in each phase. Each of these phases is different from others only in the number of common elements shared in the website.

In the Phase 1, the layout of the website is as follows.



Each web page is mainly divided into five components, top bar, left bar, right bar, bottom bar, and central content. In this phase all the web pages in our website shared the common left, top, right, and bottom bars; each page had a different content.

The top bar mainly consists of the logo of the website, search bar, and the member login. The left bar consist of navigational controls that will be used through out all the web pages for switching between pages. The right bar consists of advertisements that need to be displayed across all the web pages. The bottom bar consists of few advertisements and some other components that will be displayed across all the pages at the bottom of the page such as top headlines of the day, top picture of the month, etc.

The following is the snapshot of four pages created in the website, which will be used to test our caching mechanism.



Page 1: Home Page



Page 2



Page 3



Page 4

On close observation, it can be seen that only the central content changes in all the above web pages. The top bar, left bar, bottom bar, right bar are saved as elements as these are commonly displayed across all the web pages. Therefore, these common portions are stored as elements and are included in our web pages using our new element function. This reduced a lot of coding for the web page as we need not repeat the HTML code of these elements in every page does not need to be repeated. Instead, a single function call is done to include them.

The total response times of the web pages has been recorded using YSlow by making the requests in a LAN setup and are given below.

Page No	Response time of the web pages, developed using our new caching mechanism	Response time of the web pages, developed using the conventional approach
Page 1 – Home Page	8.327 sec	8.88 sec
Page 2	7.71 sec	9.93 sec
Page 3	7.98 sec	8.64 sec
Page 4	7.70 sec	8.73 sec

Observing the values above, notice that in requesting the initial page the difference between the responses times using our caching mechanism and conventional approach is very small; however, in our approach since the elements get cached in the browser and are served from the client instead of getting it from the actual web server in

the further requests, therefore we notice that there is a significant amount of reduction in the response times of pages 2, 3, and 4 which means we have successfully achieved our goal as to produce a reduction in response time. The maximum reduction we have obtained using our new caching mechanism is $22 \pm 0.676 \%$.

Even though we have achieved a maximum reduction of 22% in response time of the web pages, we have also observed the behavior of our caching by changing the layout of the web pages and having more elements in the page. In Phase 2, we have used the following layout for the four web pages.

Top Bar		
Left Bar	Content1	Right Bar
Advertisement section 1		
Special Content Area	Content 2	Advertisement section 2
Bottom Bar		

The entire page is divided into eight sections. All the four web pages will have the same top bar, left bar, right bar, special content area, bottom bar, advertisement section 1

and 2. However, the web pages will have different content1 and content 2 sections. All the common sections will be saved as elements in CakePHP. Therefore, we will have seven elements in this phase.

These web pages are carefully created both using our new technique and also using the conventional manner, according to the Phase 2 layout and their response times are recorded by requesting the web pages using the LAN setup and are given below.

Page No	Response time of the web pages, developed using our new caching mechanism Javascript on	Response time of the web pages, developed using the conventional approach Javascript off
Page 1 – Home Page	7.62 sec	8.13 sec
Page 2	7.99 sec	8.11 sec
Page 3	7.83 sec	7.92 sec
Page 4	7.67 sec	8.03 sec

From the results we can conclude that even with the increase in the number of elements being used in the web pages, there is definitely decrease in the response times when using our new caching mechanism.

The very idea of changing the layout and number of elements of the web pages in each phase is to observe how our caching mechanism works under different scenarios. Therefore, in Phase 3, we further broke down the page into smaller sections and the layout used is as shown below.

Advertisement Section		
Logo	Search Bar	Member Login
Navigation links	Content1	Advertisement Section 1
Special Content Area 1		Special offers Area
Advertisement section 2		
Special Content Area 2	Content 2	Advertisement section 3
Special Content Area 3		Special Content Are 4
Bottom Bar		

Each of the web pages is divided into sixteen small sections, but they differ only in the content area, i.e., they have same content in fourteen other sections. These fourteen common portions are stored as elements. The web pages are carefully created using the layout as shown in Phase 3 using both our new technique and also in conventional manner and their response times were recorded using a LAN setup and are as shown below.

Page No	Response time of the web pages, developed using our new caching mechanism	Response time of the web pages, developed using the conventional approach
Page 1 – Home Page	7.91 sec	8.03 sec
Page 2	7.74 sec	8.02 sec
Page 3	7.31 sec	7.52 sec
Page 4	7.43 sec	7.92 sec

From the results, shown above, we can observe that even with increasing the number of elements in the page, there is still as minute reduction in the response times when using our new caching mechanism.

In our final phase, we further divided the web page into thirty two smaller sections; however, they differ only in the central content area. That is the pages have thirty small portions in common. These common portions are stored as elements and the

web pages are created using our new element function. The web pages have been created using the layout in Phase 4 both using our new technique as well as using the conventional technique and their response times recorded by requesting the pages in a LAN setup and are shown below.

Page No	Response time of the web pages, developed using our new caching mechanism Javascript on	Response time of the web pages, developed using the conventional approach Javascript off
Page 1 – Home Page	14.22 sec	13.68 sec
Page 2	7.82 sec	7.83 sec
Page 3	8.07 sec	7.93 sec
Page 4	8.32 sec	8.06 sec

After recording the above results, we studied why our caching system did not produce a reduction in response times of the web pages. This revealed that even though all these common portions are stored in the client browser, the Firefox caching mechanism the web browser is taking a significant amount of time to get the content from its own cache, which seemed to be the main reason as to why there was no improvement in the response time of the web pages.

As the above experiments involved analyzing how our caching system works on a number of elements in a page. We conducted several tests to witness how our caching system works as the size of the elements increase. Therefore, we have designed

web pages that had five sections. Each page consisted of four elements which were shared among all the pages; however, they had different content.

This experiment was done in several stages. In each stage, the four web pages had different content but shared common elements. The size of the elements started with 20k and was incremented to 200k with an increment of 20k at each stage. The html files of the four pages in the stage one are shown below.

Page Number	HTML Code
Page 1	<pre> <p> <!--Content of the page 1 -- > </p> <!--element1--> <p> <?php echo AddonsView::element('element1_20k',array("cache"=>"5 min"), null, \$status) ?> </p> <!--/element1--> <!--element2--> <p> <?php echo AddonsView::element('element2_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element2--> <!--element3--> <p> <?php echo AddonsView::element('element3_20k',array("cache"=>"5 min"), null, \$status); ?> </p> </pre>

	<pre> <!--/element3--> <!--element4--> <p> <?php echo AddonsView::element('element4_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element4--> </pre>
Page 2	<pre> <p> <!--Content of the page 2 -- > </p> <!--element1--> <p> <?php echo AddonsView::element('element1_20k',array("cache"=>"5 min"), null, \$status) ?> </p> <!--/element1--> <!--element2--> <p> <?php echo AddonsView::element('element2_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element2--> <!--element3--> <p> <?php echo AddonsView::element('element3_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element3--> </pre>

	<pre> <!--element4--> <p> <?php echo AddonsView::element('element4_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element4--> </pre>
Page 3	<pre> <p> <!--Content of the page 3 -- > </p> <!--element1--> <p> <?php echo AddonsView::element('element1_20k',array("cache"=>"5 min"), null, \$status) ?> </p> <!--/element1--> <!--element2--> <p> <?php echo AddonsView::element('element2_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element2--> <!--element3--> <p> <?php echo AddonsView::element('element3_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element3--> <!--element4--> </pre>

	<pre> <p> <?php echo AddonsView::element('element4_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element4--> </pre>
Page 4	<pre> <p> <!--Content of the page 4 -- > </p> <!--element1--> <p> <?php echo AddonsView::element('element1_20k',array("cache"=>"5 min"), null, \$status) ?> </p> <!--/element1--> <!--element2--> <p> <?php echo AddonsView::element('element2_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element2--> <!--element3--> <p> <?php echo AddonsView::element('element3_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element3--> <!--element4--> <p> <?php echo </pre>

	<pre>AddonsView::element('element4_20k',array("cache"=>"5 min"), null, \$status); ?> </p> <!--/element4--></pre>
--	------------------------------------------------------------------------------------------------------------------------------------

On closing observing the above four web pages you can find that the web pages only differ in the content. In this manner, as stages proceed we increase the size of the element by 20k; however, only the content will be different in the web pages at a particular stage.

We conducted this experiment phase by phase by requesting the web pages from the client, both using our new caching mechanism as well as the conventional way.

Following tables list the response times of the web pages recorded using YSlow, a Firefox add-on. Table 1 represent the response times of web pages developed using our new caching mechanism and Table 2 represents the response times of the web pages developed using the conventional method.

Table 1: Response Times recorded using our caching mechanism

Element Size	Response Time of Page 1 (in sec)	Response Time of Page 2 (in sec)	Response Time of Page 3 (in sec)	Response Time of Page 4 (in sec)
20k	9.85	7.245	7.43	7.38
40k	11.575	7.065	7.06	7.2465
60k	11.445	7.175	7.285	7.235
80k	11.75	7.85	8.01	7.67
100k	11.01	7.62	7.53	7.49

120k	10.55	7.83	7.01	7.63
140k	11.41	7.88	7.95	7.97
160k	10.21	8.13	8.7	8.15
180k	11.24	8.38	8.4	8.23
200k	10.35	8.26	8.72	8.56

Table 2: Response Time using conventional mechanism

Element Size	Response Time of Page 1 (in sec)	Response Time of Page 2 (in sec)	Response Time of Page 3 (in sec)	Response Time of Page 4 (in sec)
20k	11.77	9.953	8.57	8.5
40k	8.98	9.13	8.85	8.652
60k	8.64	8.004	8.43	7.92
80k	8.63	8.47	8.87	8.41
100k	7.99	8.35	8.23	7.89
120k	8.75	7.92	7.96	8.12
140k	8.5	8.34	8.23	9.03
160k	8.83	8.54	9.01	8.67
180k	8.1	9.07	8.96	9.06
200k	9.13	9.01	9.32	9.2

The following graph is calculated to analyze how much percentage increase was made using our new caching mechanism.

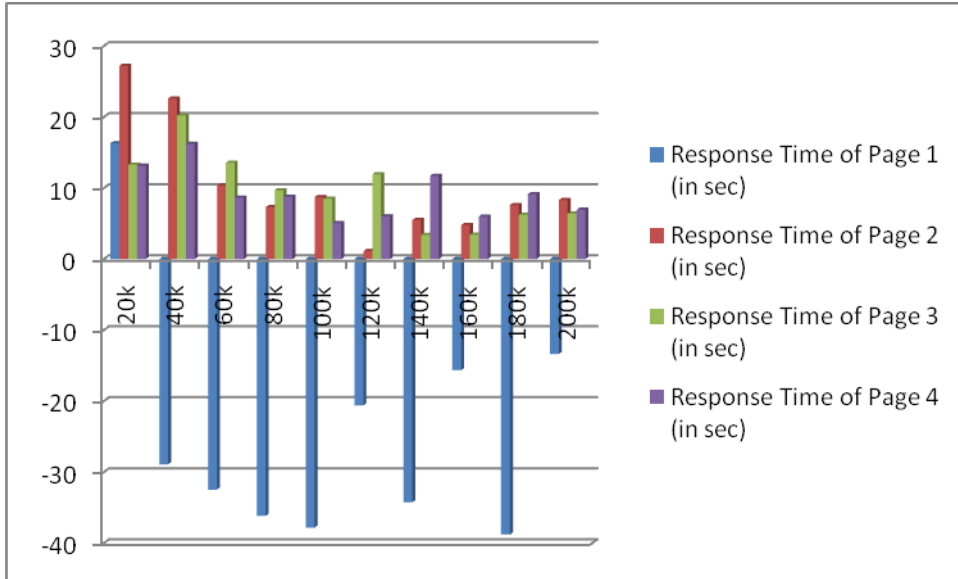


Figure 11: Results obtained using Firefox browser

From the above graph we notice that, even though the response time of a web Page 1 is more when using our new caching mechanism versus using the conventional method, for Pages 2, 3, and 4, even with the increase in the size of the element, there is definitely an improvement in the response time using our caching technique. The maximum amount of reduction produced using our new caching mechanism is $27 \pm 0.417\%$.

We have also conducted the above experiment by requesting pages using Internet Explorer and the following graph represents the percentage difference in the web page response times of the web pages, recorded using iMacros.

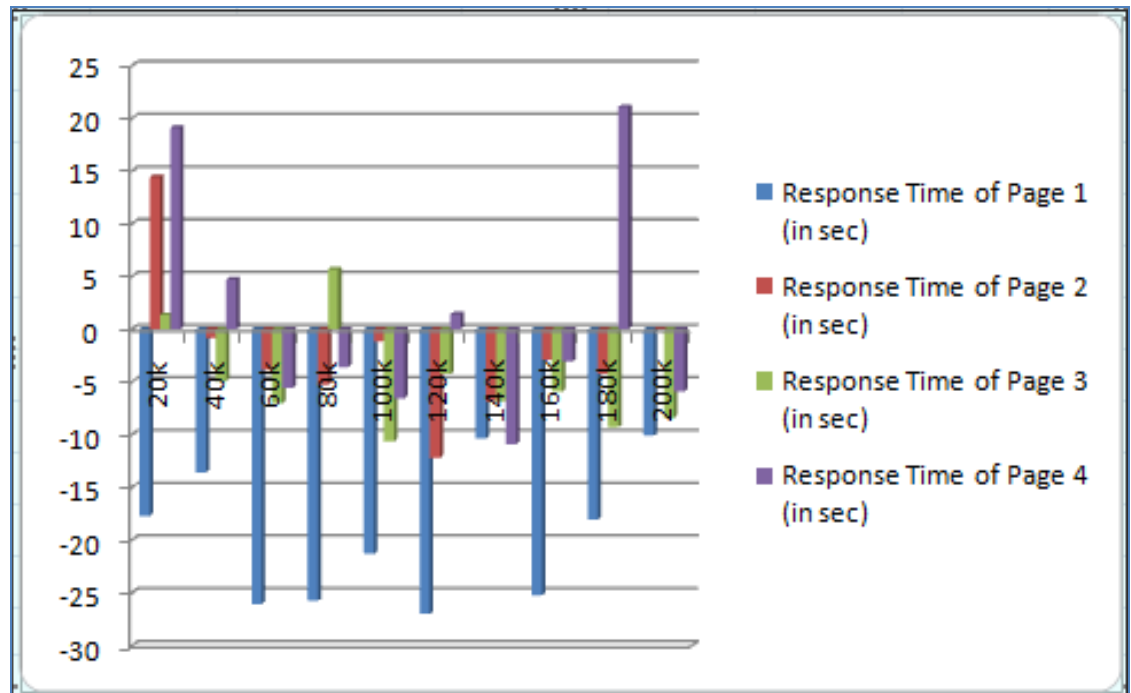


Figure 12: Results obtained using Internet Explorer

It can be observed from the above graph that, using our new caching mechanism the response times of the web pages was greater than the response times of the web pages recorded using the conventional mechanism. On closing observing the results, we have noticed that Internet Explorer takes more than 500 – 600 milliseconds to get the elements from its own cache which is more than what is needed to get the original text from the server, therefore there was no improvement found while using Internet Explorer.

In order to observe the behaviour of our caching system using Opera browser, we do not have a YSlow plug-in or a iMacros plug-in, therefore, we have used CURL, a command line tool for transferring files, to record the total download times of the web pages. The following was the graph obtained.

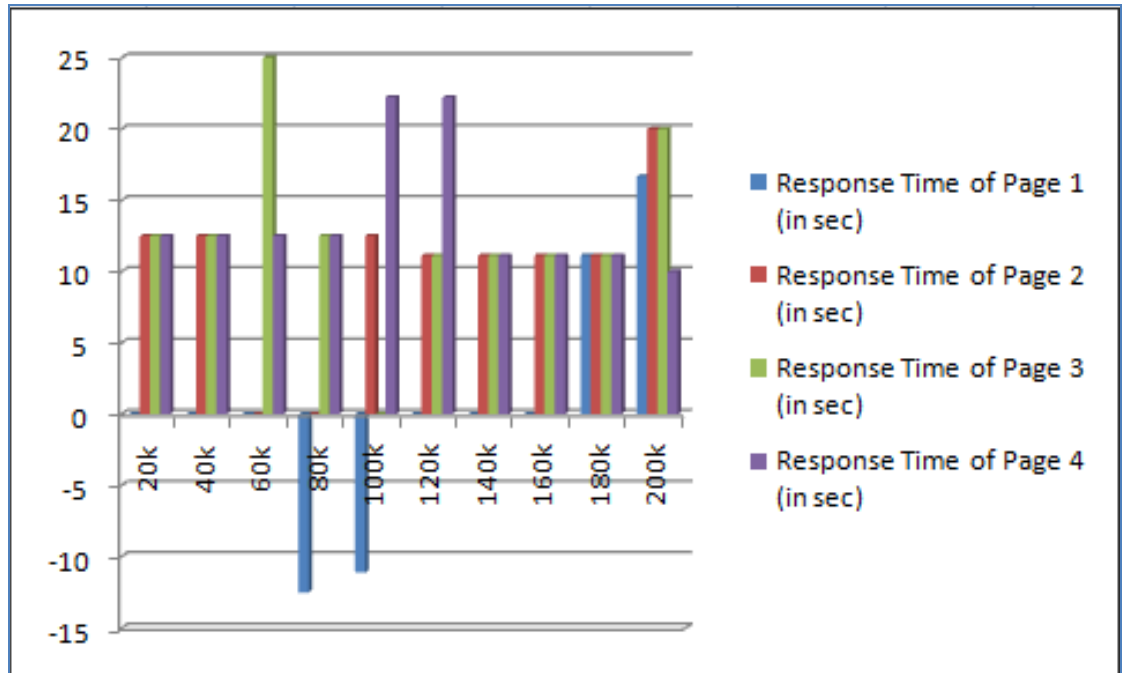


Figure 13: Results obtained using Opera

The above graph, displays the percentage difference between the response time of the web pages measured using our caching mechanism and using the conventional method. We can notice from the graph that our caching system has an advantage over the conventional approach when the pages are requested using Opera browser. The maximum improvement that has been recorded is $25 \pm 0.589\%$.

We have also tested our caching system in Safari browser. As Safari also did not have any YSlow or iMacros plug-in, we have used a similar approach used for testing our mechanism in Opera browser i.e., we have used CURL to record the response times of the web pages. The values shown in the below graph represent the percentage difference between the response times of the web pages recording using our new caching system and the conventional system.

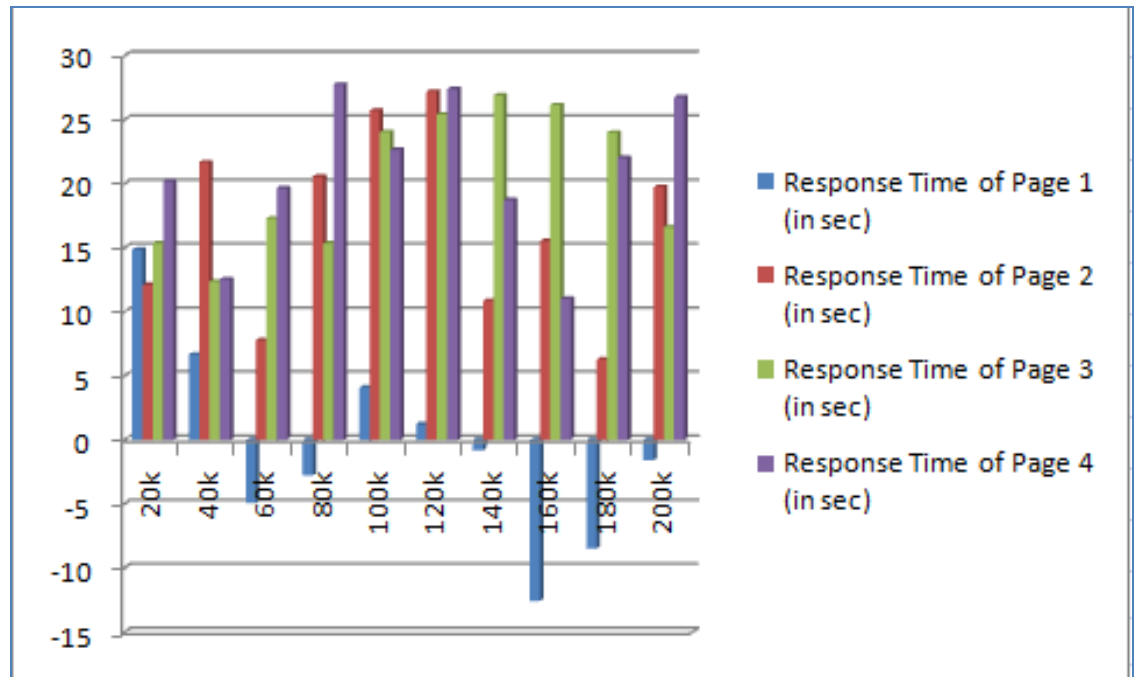


Figure 14: Results obtained using Safari browser

From the above graph, we can conclude that our caching mechanism definitely has an advantage over the conventional technique in displaying the web pages and the maximum percentage increase obtained while using Safari browser is $27 \pm 0.871\%$

5.3.4 Remarks

Testing our caching mechanism showed that it was possible to produce a reduction in the response time by caching the common portions of the website and reusing them in the further requests. However, while we were recording our values using YSlow, we noticed that in order to get elements from its own cache, the Firefox web browser took few hundred milliseconds to get each element. All the above values in Table 1 include the time taken to get the entire page, therefore if this time to get the

cached elements gets reduced, we can get even more reduction in response times of the web pages.

On delving into the cause for this issue, we noticed that Firefox used memory cache to store the images in its cache, whereas it used disk cache to store the Javascripts and CSS files. Therefore, to retrieve images from its cache, it needs only a few milliseconds, whereas in order to get the Javascripts to fetch the element from the disk it takes a few hundred milliseconds. If Firefox web browser can also store our external Javascripts which are our actual elements in its memory cache, we will be able to retrieve these elements much faster and a huge reduction in response times of the web pages can be obtained.

On observing how our the caching mechanism works with Internet Explorer, we have observed that using this new mechanism for displaying the web pages does not provide any advantage as Internet Explorer requires at-least 500 to 600 ms to get the elements from its own cache. Whereas, from the experiments conducted using Safari and Opera, we can conclude that we have huge advantage using our new caching system over the conventional mechanism i.e., caching the most common portions of the websites at the client-side and reusing them in further requests has definitely increases the productivity of the user as the pages get displayed faster. The maximum percentage increase recorded was using Safari browser and the value is $27 \pm 0.871\%$

6 Conclusion

Identifying the most common portions of the website and caching them at the client so as to reuse them in the further requests is definitely a new direction as to how web caching is currently being done. From our experiments, we have observed that we can obtain at least 20% improvement in the response times of the web pages when requested using Firefox, Safari, and Opera browsers having less number of common elements shared on the web pages and the size of the elements should be less than 60KB. However, as number of the elements increase there is decrease in the improvement of response time. This was observed because our caching mechanism was developed in combination with web browsers caching technique and CakePHP element function. Web browsers require a significant amount of time to get the elements from their cache. This issue can definitely reduced by creating an efficient procedure that store and retrieve the external Javascripts on the client-side. If this time to get the elements from the browsers cache is improved, then developing the web pages using our caching technique will have a performance advantage.

7 References

Best Practices for Speeding up Your Web Site. Retrieved September 27, 2008, from

<http://developer.yahoo.com/performance/rules.html>

CakePHP. Retrieved November 12, 2008, from <http://cakephp.org/>

Cevasco, F. (2006). The CakePHP Framework: Your First Bite. Retrieved July 12, 2006,

from <http://www.sitepoint.com/article/application-development-cakephp>

PHP. Retrieved November 12, 2008, from <http://www.php.net/>

Proxy Server, Retrieved December 8, 2008, from

http://en.wikipedia.org/wiki/Proxy_server

Speed up your web pages with YSlow. Retrieved April 11, 2009, from

<http://developer.yahoo.com/yslow/>

Squid: Optimizing Web Delivery. Retrieved November 13, 2008, from

<http://www.Squid-cache.org/>